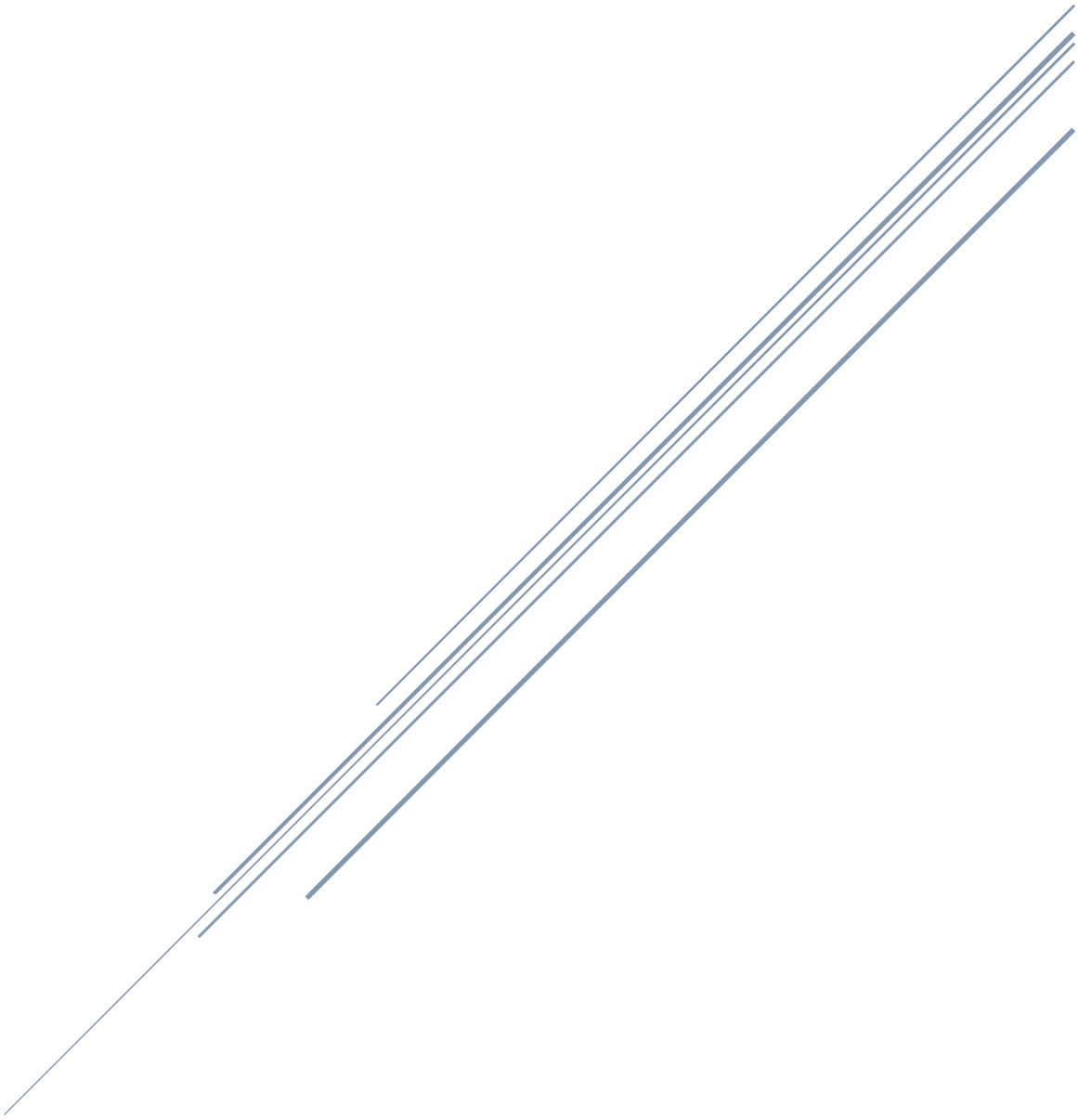
PLANIFICACIÓN Y DE SIMULACIÓN DE REDES

Grupo 04



Curso 15/16

Índice

1. Introducción

El escenario que se presenta es un sistema en el que existe un único servidor que atiende todas las peticiones de los clientes, puesto que no es un servidor con muchos accesos simultáneos y un único servidor puede atender a todas las peticiones sin perjudicar en exceso la QoS, como podemos observar en la *imagen 1*.

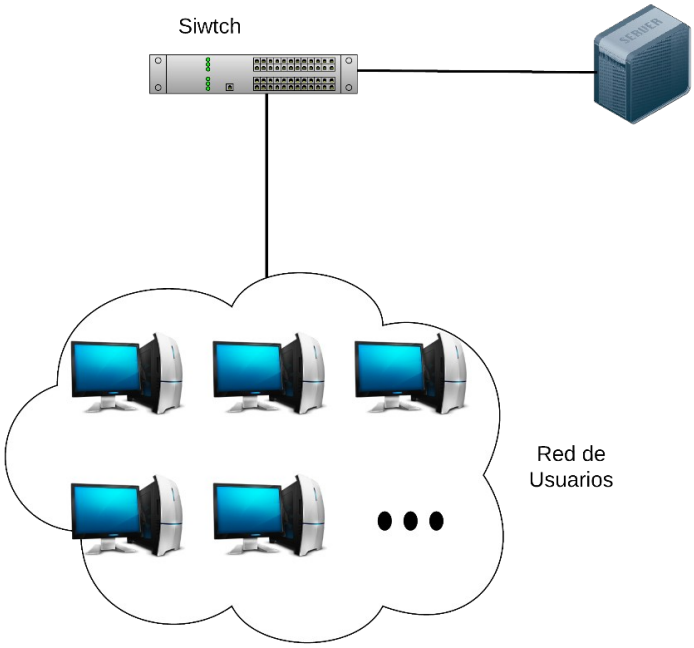


Imagen 1. Sistema Actual

El problema surge cuando comienza a incrementarse el número de accesos al servidor, los cuales congestionan mucho la red y perjudican considerablemente la QoS, empeorando así la experiencia de los clientes.

Por ello nace la necesidad de modificar el sistema, de forma óptima, en base al número de usuarios que van a acceder, en media, al contenido del servidor. Así, este proyecto trata diferentes implementaciones para comprobar la mejor forma de abordar la implementación del sistema para mantener una buena QoS sin derrochar demasiados recursos, pero dejando un margen de recursos libres para tratar un aumento de accesos futuros.

Para dotar al sistema de un Balanceador de Carga se ha decidido hacer uso del concepto de Redes Definidas por Software (SDN, por sus siglas en inglés). Estas redes se basan principalmente en el protocolo OpenFlow. Este protocolo rige el intercambio de mensajes entre los elementos de red y el Controller, para que de esta manera las órdenes y el intercambio de información entre todos los dispositivos se realice de manera adecuada y acorde al estándar.

2. Estructura y contexto del proyecto

Como podemos comprobar, el escenario desde el que vamos a partir es un único servidor que desborda ante el aumento de accesos al mismo. Por lo que como solución vamos a añadir nuevos servidores y en el switch que intercomunica a clientes y servidores vamos a hacer uso de OpenFlow para la comunicación del mismo con un controller que decida como redirigir ese mensaje. Esta redirección se va a realizar a nivel de enlace, por lo que el switch sólo va a tener dirección IP en la interfaz pública, para que de cara a los clientes siga existiendo un único servidor y el balanceo se haga de forma transparente a los mismos.

2.1 Arquitectura SDN

Las redes definidas por software (SDN) son un conjunto de técnicas que tienen como objetivo hacer una red lo más centralizada posible y evitando en la medida de lo posible a la inclusión de equipos físicos en la red. Así, este tipo de redes aísla el dispositivo físico de la lógica que se realiza.

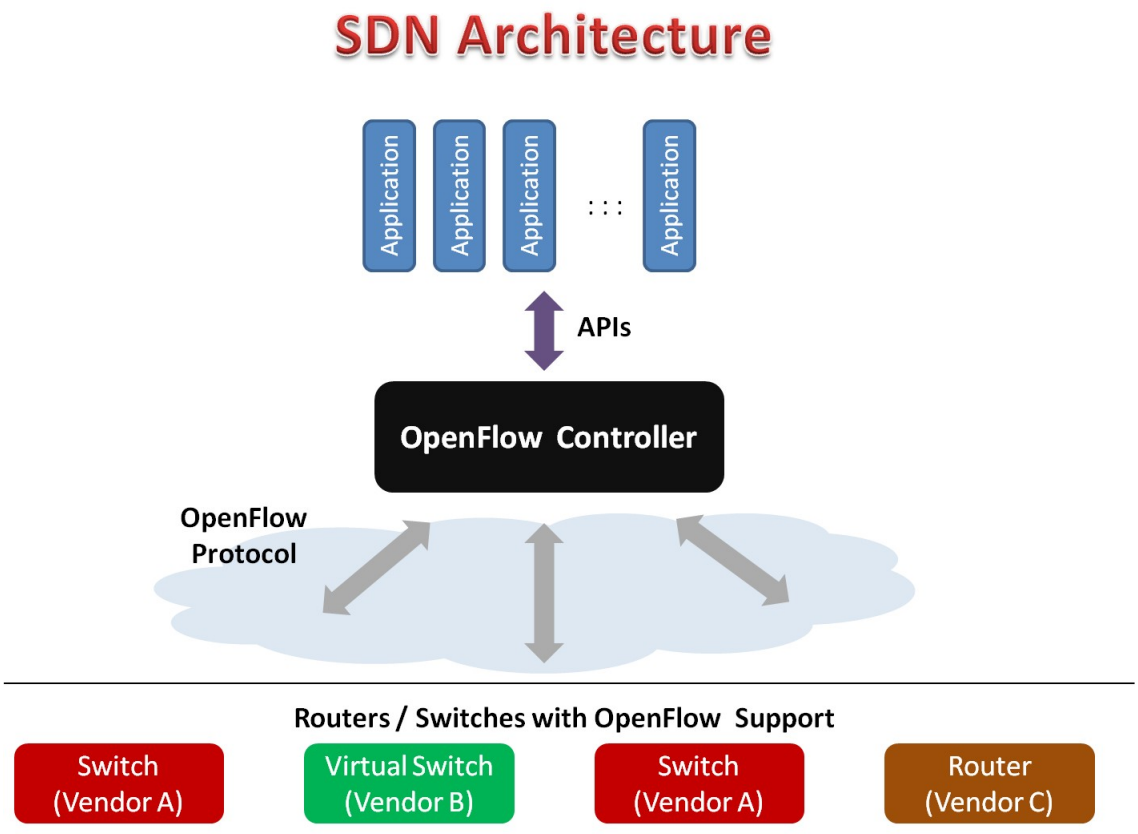


Imagen 2. Arquitectura SDN

2.2 Openflow

OpenFlow es un protocolo de comunicación entre el controlador SDN (controller) y los dispositivos conectados a éste. Su objetivo es la comunicación lógica/datos entre ambas partes indicándole al dispositivo la ruta de reenvío que debe seguir. En este proyecto este reenvío se hace a nivel de enlace, enviando por los diferentes puertos del switch a los que se encuentran conectados los servidores.

En la siguiente imagen podemos observar el intercambio de mensajes que se realizan desde que el cliente intenta acceder al servidor, hasta que el switch, mediante la consulta al controller, se comunica con el servidor correspondiente:

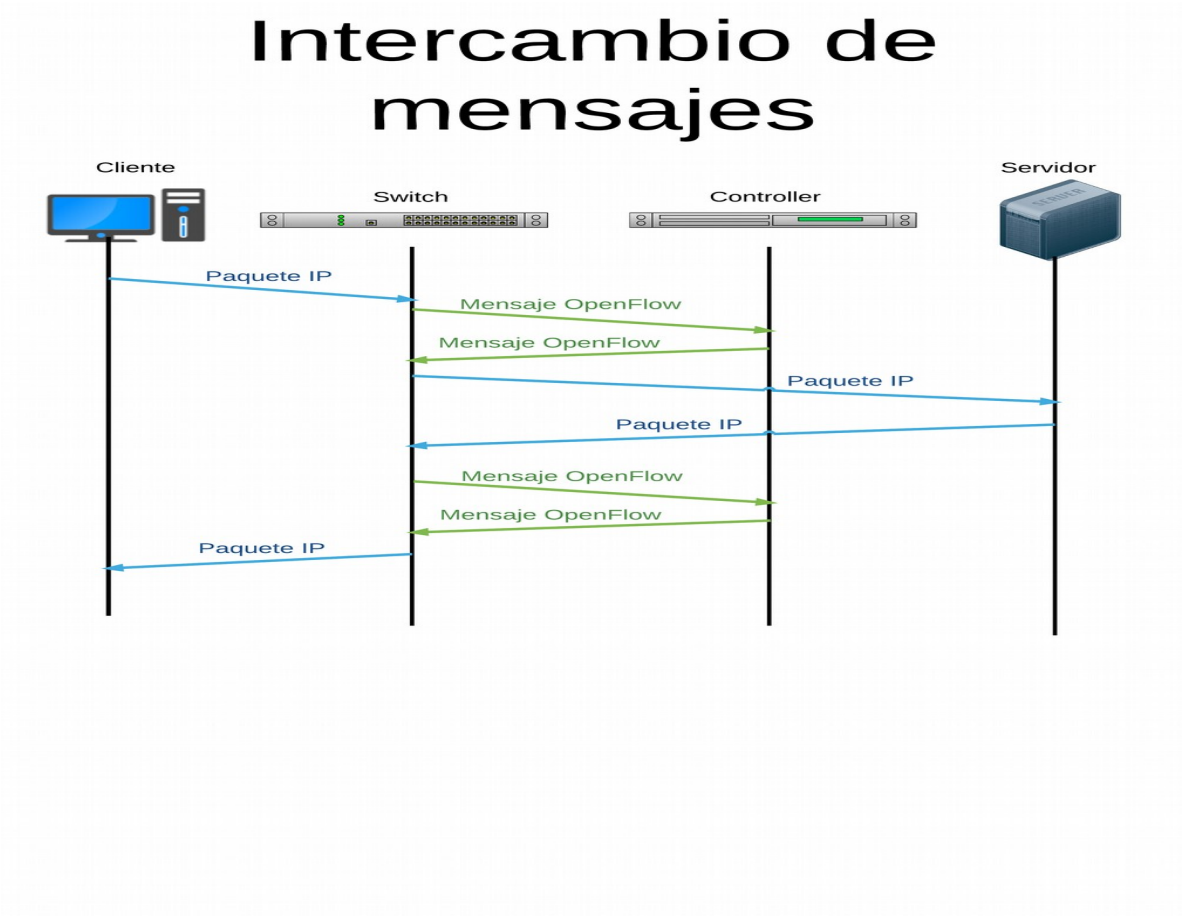


Imagen 3. Paso de mensajes entre cliente y servidor

En este escenario vamos a utilizar un switch híbrido, cuya funcionalidad es la mezcla de OpenFlow con protocolos de la capa 2/3 del modelo TCP/IP, ya que el switch utiliza OpenFlow en la subred privada, mientras que hace uso de los protocolos de la capa de red para comunicarse con los clientes, lo que compone la subred pública.

2.3 Controller

Es el dispositivo encargado de llevar a cabo la lógica de conexión entre el switch y los diferentes servidores. Su función consiste en balancear, según los diferentes algoritmos que posteriormente se van a comentar, el tráfico que llega desde los clientes a la IP de la interfaz pública del switch por un determinado puerto del mismo para así llegar a nivel de enlace al servidor correspondiente y no saturar ningún servidor. En la *imagen 4* podemos ver el funcionamiento general del controller, tras la realización de una petición por parte de un cliente al servidor.

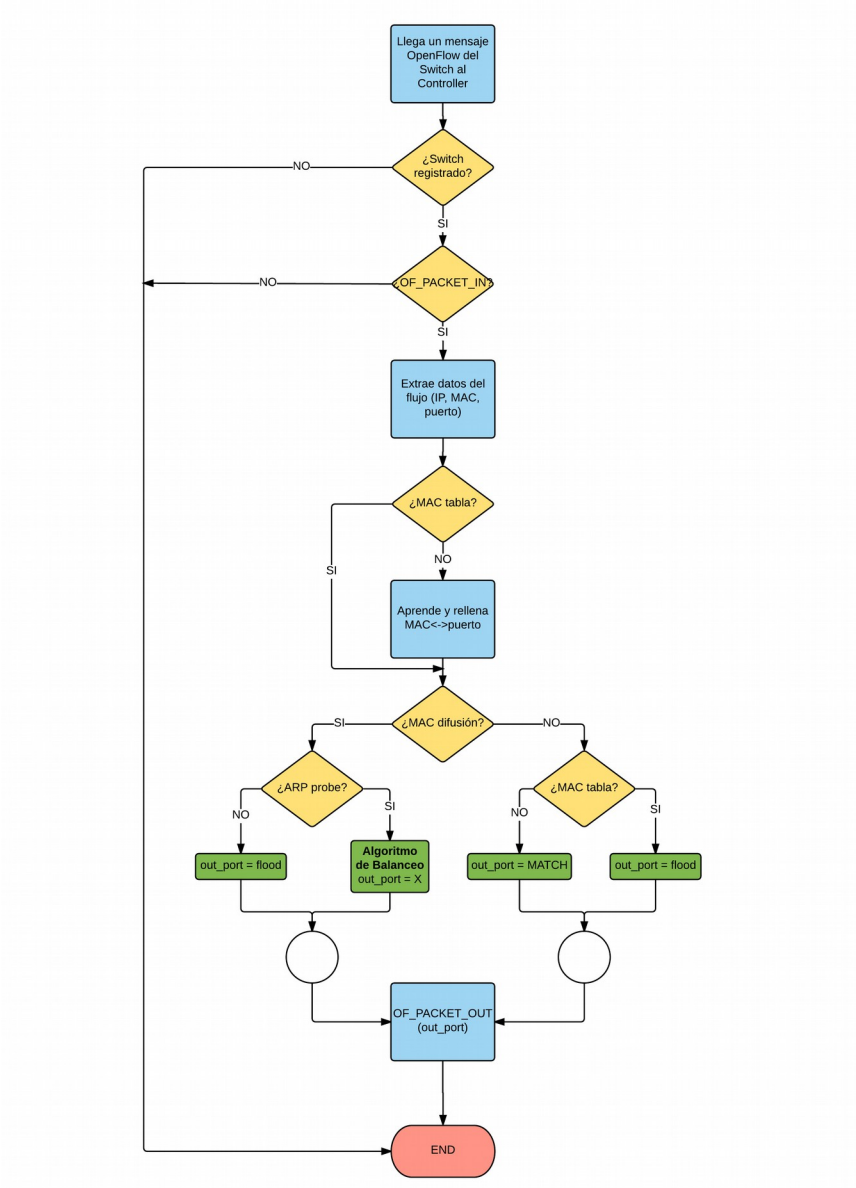


Imagen 4. Funcionamiento general del controller

3. Desarrollo de soluciones

De cara a abordar el problema surgido, vamos a hacer uso del simulado ns-3 para realizar la simulación en un entorno cerrado de tres posibles

métodos diferentes de balanceo para comparar las soluciones obtenidas y definir cuál es el mejor método para el sistema implementado modificando el número de clientes que realizan peticiones al sistema, el número de servidores que componen el sistema, la capacidad del canal, el retraso del mismo y el tipo de algoritmo utilizado.

El sistema se divide principalmente en tres partes, que se pueden observar visualmente en la *imagen 5*:

- Clientes: son los encargados de realizar peticiones a los equipos servidores, que desde el punto de vista de los clientes sólo existe un servidor, que es la dirección del switch. Se encuentran conectados directamente al switch.
- Switch o router: es el dispositivo encargado de encaminar las peticiones que le llegan desde su interfaz pública a nivel de red, cuya dirección IP es la 172.16.0.1, por los diferentes enlaces que comunican con los servidores, todo ello realizando la consulta a un controller para saber el enlace por el que reenviar. Es el dispositivo que interconecta clientes conservidores.
- Servidores: son los encargados de procesar y responder a las peticiones que realizan los equipos clientes. Se encuentra conectado directamente al switch.

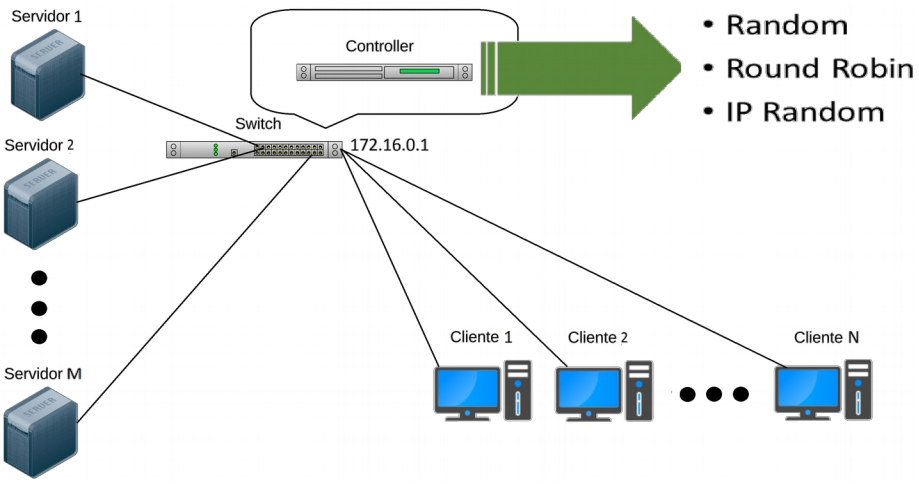


Imagen 5. Estructura del sistema modificado

En este sistema el controller es el encargado de decidir cómo reenviar los paquetes a los servidores y va a tener los siguientes algoritmos de balanceo de carga:

- Random
- Round Robin
- IP Random

Estos algoritmos los vamos a desarrollar en mayor profundidad en los siguientes puntos, pero su principal objetivo es encontrar el algoritmo

óptimo de balanceo para conservar la mayor QoS posible, pero sin derrochar recursos.

3.1 Algoritmo RANDOM

En este modo las peticiones se balancean entre los diferentes servidores de manera aleatoria. Cuando llega una petición al switch este determina a qué servidor la envía de manera totalmente aleatoria, para ello se genera un número aleatorio uniforme entre 0 y el número de servidores. Tras un número suficientemente grande de iteracciones la carga debe ser uniforme en todos los servidores, puesto que estadísticamente todos los servidores tienen la misma probabilidad de recibir una petición. Podemos observar gráficamente el comportamiento de este algoritmo en la *imagen 6.*

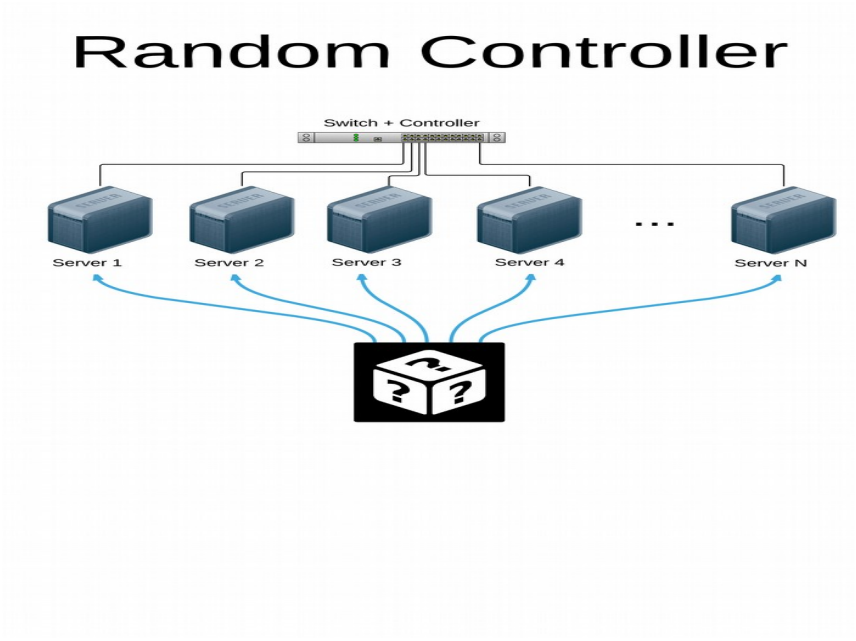


Imagen 6. Balanceo con el algoritmo random

3.2 Algoritmo ROUND ROBIN

En este modo las peticiones se balancean entre los diferentes servidores de mediante el protocolo Round Robin. Cuando llega una petición al switch este determina a qué servidor la envía de manera secuencial, para ello almacena en una lista el último servidor al que envió la petición para en el caso actual mandarla al siguiente, cuando llega al final de la lista vuelve a enviarla al primero. La carga debe ser uniforme en todos los servidores sin tener que esperar a que se realice un número grande de simulaciones, puesto que no depende de ningún factor aleatorio, simplemente asigna de manera circular las peticiones. Podemos observar gráficamente el comportamiento de este algoritmo en la *imagen 7.*

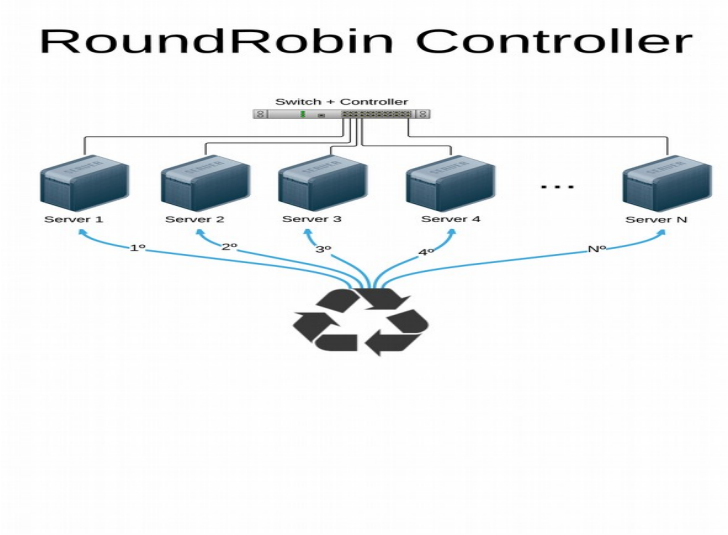


Imagen 7. Balanceo con el algoritmo round robin

3.3 Algoritmo IP RANDOM

En este modo las peticiones se balancean entre los diferentes servidores en dos fases. Primero teniendo en cuenta la IP origen se determina si ésta es par o impar, en caso de que la IP sea par, la petición será atendida solo por servidores pares. En caso de ser impar se atiende por servidores impares. Una vez determinado esto se genera aleatoriamente el servidor que la atiende, es decir, dentro de los pares o los impares se determina cuál es el que atiende la petición. Tras un número suficientemente grande de iteraciones la carga debe ser uniforme en todos los servidores, puesto que estadísticamente todos los servidores tienen la misma probabilidad de recibir una petición debido a que el tráfico real tendrá el mismo número de IPs pares que impares, y dentro de cada grupo hay la misma probabilidad de elegir un servidor u otro dentro de la lista de servidores pares o impares. Podemos observar gráficamente el comportamiento de este algoritmo en la *imagen 8.*

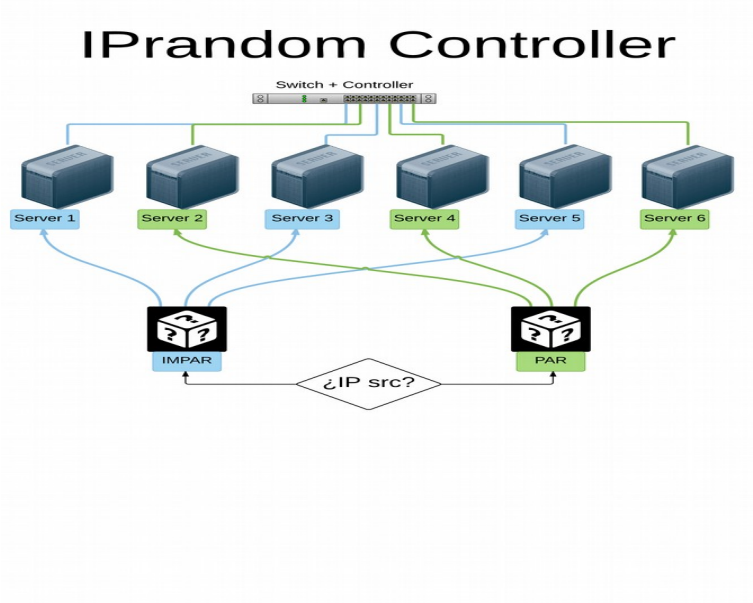


Imagen 8. Balanceo con el algoritmo IP Random

4. Código del proyecto

Previamente a la creación del código necesario para conseguir el propósito del proyecto y sus posteriores pruebas debemos realizar la instalación de OpenFlow en nuestro equipo e integrarlo con ns-3. Dicha instalación se encuentra detallada en el *Anexo A*.

Para la resolución del problema que se nos presenta y partiendo de un fichero principal, se crean una serie de ficheros encargados de definir los diferentes algoritmos que tratan el problema. Además se crea un observador para cada servidor para que muestre las diferentes estadísticas, en función del método de balanceo escogido, número de clientes y número de servidores.

A continuación vamos a detallar los diferentes ficheros utilizados en este proyecto, explicando su funcionamiento principal y para qué sirven.

4.1 Topología.cc

Fichero principal que engloba toda la simulación. Se divide en diferentes métodos:

- Simulación: engloba la creación del escenario que vamos a tener, así como la creación de trazas para comprobar el tiempo entre paquetes recibidos en media entre todos los servidores del sistema. Además es el encargado de comenzar y finalizar la simulación.
- Main: trata la creación de las variables que se van a utilizar a lo largo del programa y obtiene a partir de la línea de comandos variables como el número de servidores, el algoritmo a utilizar, la tasa de bits o el retardo de propagación. Además crea el controller concreto en base al algoritmo elegido y genera las diferentes

gráficas para mostrar visualmente qué resultados son mejores y en qué casos.

4.2 Controller.h

Fichero encargado de definir los controladores de los diferentes algoritmos que forman parte del sistema, así como los métodos que forman parte de los mismos y sus variables.

4.3 Loadbalancer.h

Fichero que contiene los diferentes tipos de balanceo existentes. Además posee un tipo añadido que indica si hay algún error. Todo ello se encuentra definido en un elemento de tipo *struct*.

4.4 Random-controller.cc, Roundrobin-controller.cc, Iprandom-controller.cc

Este conjunto de ficheros contienen la lógica del algoritmo random, round robin y ip random respectivamente. Estos ficheros además contienen diferentes trazas de tipo LOGIC para comprobar en todo momento entre que dispositivos se ha establecido la conexión, mostrando valores como MAC origen y destino e IP origen y destino. Además indica si la dirección MAC se encuentra en la tabla o la ha añadido en esa conexión.

4.5 Observador.h

Fichero que contiene la declaración de los diferentes métodos y variables que son necesarios por el fichero *observador.cc*.

4.6 Observador.cc

Fichero que obtiene los métodos necesarios para la obtención de todas las estadísticas necesarias a lo largo de la simulación.

5. Análisis de resultados

Para la realización de pruebas podemos ejecutar el proyecto que hemos realizado con parámetros adicionales o sin ningún parámetro adicional. En caso de querer ponerle parámetros adicionales, estos son:

- number: número de servidores que tendrá inicialmente el sistema. Por defecto su valor es 4.
- type: tipo de algoritmo que vamos a utilizar en el sistema. Este tipo puede ser random, round-robin o ip-random. Por defecto es el algoritmo random.
- dataRate: tasa de bits que tendrá el sistema. Por defecto su valor es 100Mbps.

- delay: retraso medio del canal. Por defecto su valor es 6.56us.

A continuación vamos a mostrar el resultado de algunas de las simulaciones realizadas explicando los resultados que hemos obtenido.

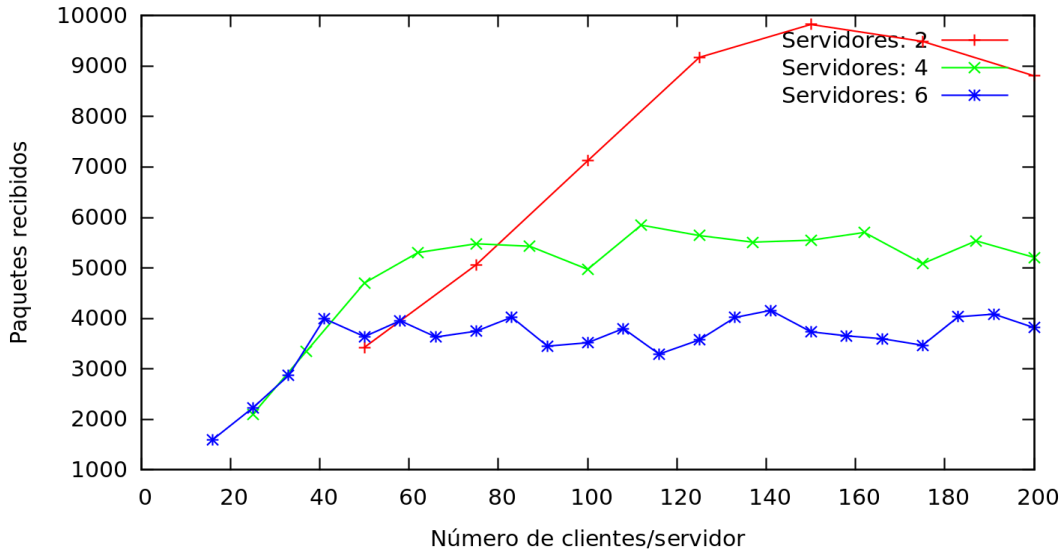
- **Balanceo de carga con algoritmo Random y variando el número de clientes**

  Datos Iniciales
  - Clientes iniciales: 100
  - Tipo de balanceo: Random
  - Servidores iniciales: 2, 4 y 6
  - DataRate: 100Mbps
  - Delay: 5us

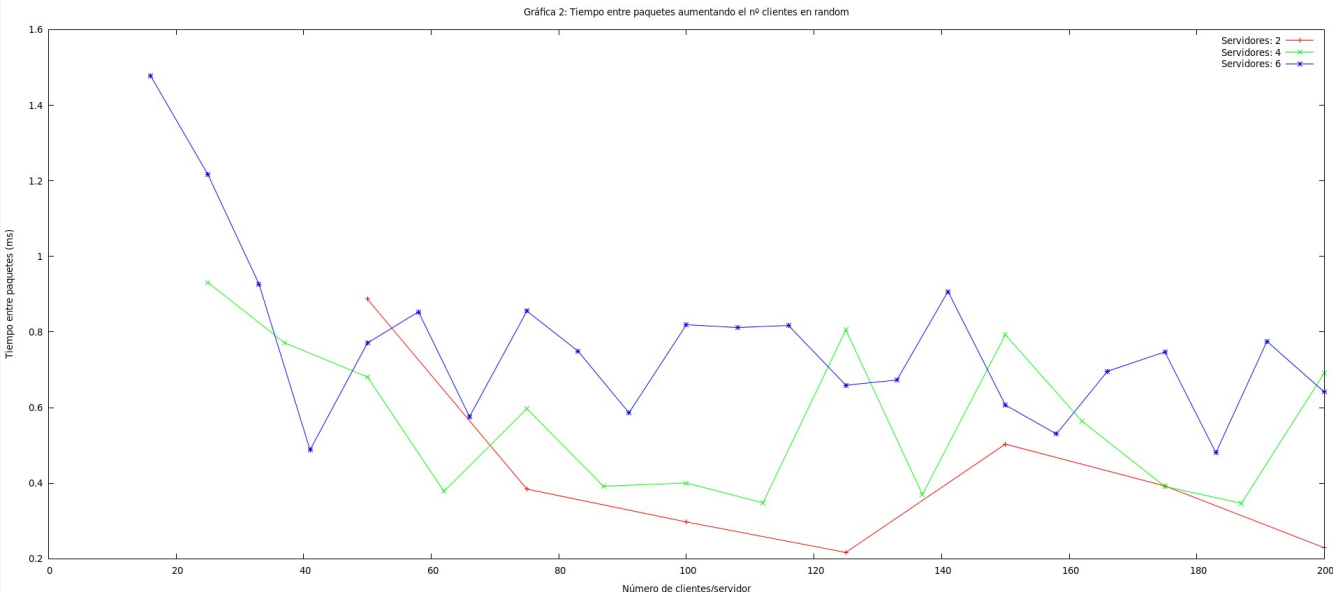
  Incrementos por punto
  - Clientes: 50

  En esta simulación vamos a observar dos gráficas: media de paquetes recibidos (*Gráfica 1*) y tiempo medio entre paquetes (*Gráfica 2*). En estas gráficas se van a representar 3 curvas, en las que tendremos 2, 4 y 6 servidores, cuyos valores son fijos, y donde vamos a utilizar un balanceo de carga de tipo random. Además se tiene un incremento de 50 clientes por cada punto.

Gráfica 1: Paquetes recibidos aumentando el nº clientes en random



Gráfica 1. Paquetes recibidos aumentando el nº de clientes en random



Gráfica 2. Tiempo entre paquetes aumentando el nº de clientes en random

A continuación procedemos a explicar brevemente los resultados de cada gráfica:

Gráfica 1:
En esta gráfica podemos observar la evolución de los paquetes recibidos, en función del aumento del número de clientes según el número de servidores hábiles.
Al aumentar el número de clientes, con 2 servidores disponibles, las peticiones recibidas en media por cada servidor, van aumentando de forma considerable.
Sin embargo, conforme aumenta el número de servidores habilitados se puede apreciar como cada servidor recibe, en promedio, menos paquetes, ya que estos son repartidos entre los distintos servidores.

Gráfica 2:
Aquí podemos observar que, el tiempo entre 2 paquetes recibidos consecutivamente, en media de cada servidor, aumenta conforme al incremento de servidores, lo cual es lógico, al haber más servidores atendiendo peticiones, cada uno tardará más en recibir un paquete.
Los distintos picos de cada curva, pueden explicarse como incrementos de peticiones simultáneas realizadas por los clientes en dicho instante.

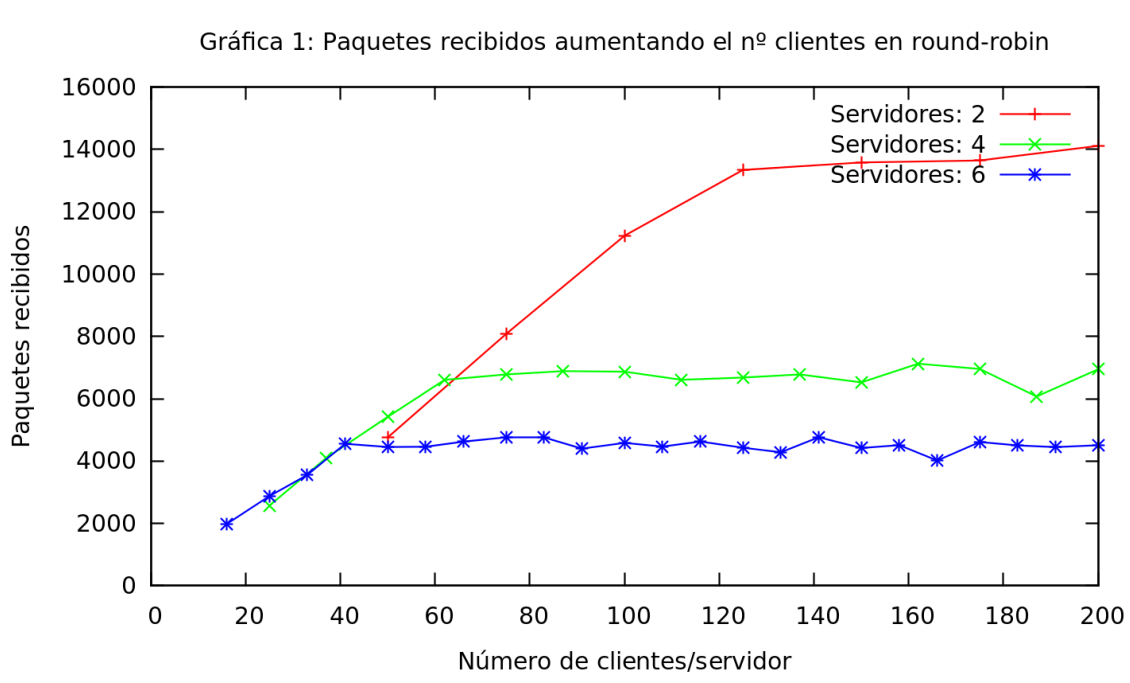
- **Balanceo de carga con algoritmo Round Robin y variando el número de clientes**

  Datos Iniciales
  - Clientes iniciales: 100
  - Tipo de balanceo: Round Robin
  - Servidores iniciales: 2, 4 y 6
  - DataRate: 100Mbps
  - Delay: 5us

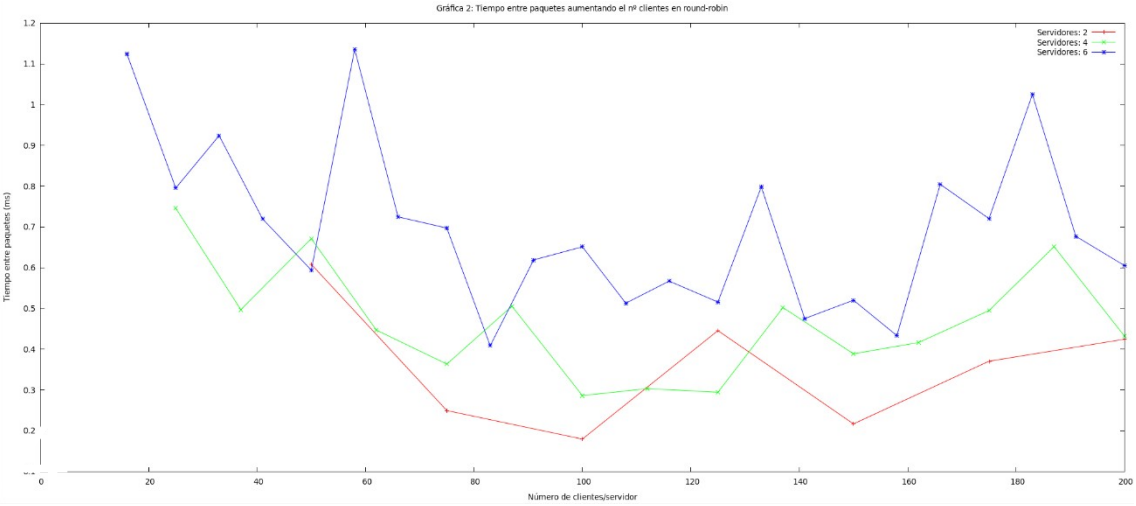
  Incrementos por punto

- Clientes: 50

En esta simulación vamos a observar dos gráficas: media de paquetes recibidos (*Gráfica 1*) y tiempo medio entre paquetes (*Gráfica 2*). En estas gráficas se van a representar 3 curvas, en las que tendremos 2, 4 y 6 servidores, cuyos valores son fijos, y donde vamos a utilizar un balanceo de carga de tipo round robin. Además se tiene un incremento de 50 clientes por cada punto.



Gráfica 1. Paquetes recibidos aumentando el nº de clientes en round robin



Gráfica 2. Tiempo entre paquetes aumentando el nº de clientes en round robin

A continuación procedemos a explicar brevemente los resultados de cada gráfica:

Gráfica 1:
En esta gráfica, al igual que en el caso anterior, podemos observar la evolución de los paquetes recibidos, conforme se incrementan el número de clientes según el número de servidores.
Al aumentar el número de clientes, con 2 servidores disponibles, las peticiones recibidas en media por cada servidor, van aumentando de forma considerable.
Sin embargo, conforme aumenta el número de servidores habilitados se puede apreciar como cada servidor recibe, en promedio, menos paquetes, ya que estos son repartidos entre los distintos servidores.

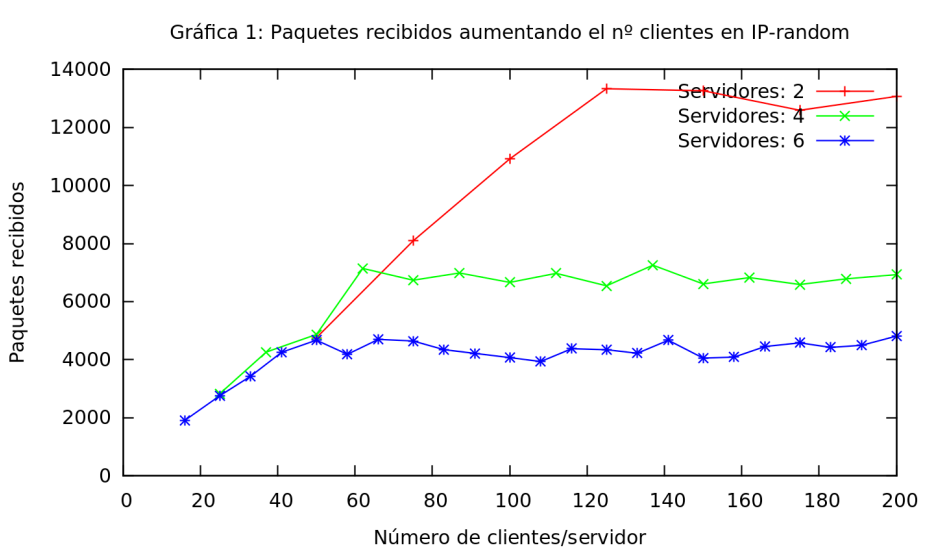
Gráfica 2:
Observamos como el tiempo entre 2 paquetes recibidos consecutivamente, en media por cada servidor, aumenta conforme se incrementa el número de servidores, ya que al haber más servidores atendiendo peticiones, cada uno tardará más en recibir un paquete.

- **Balanceo de carga con algoritmo IP Random y variando el número de clientes**

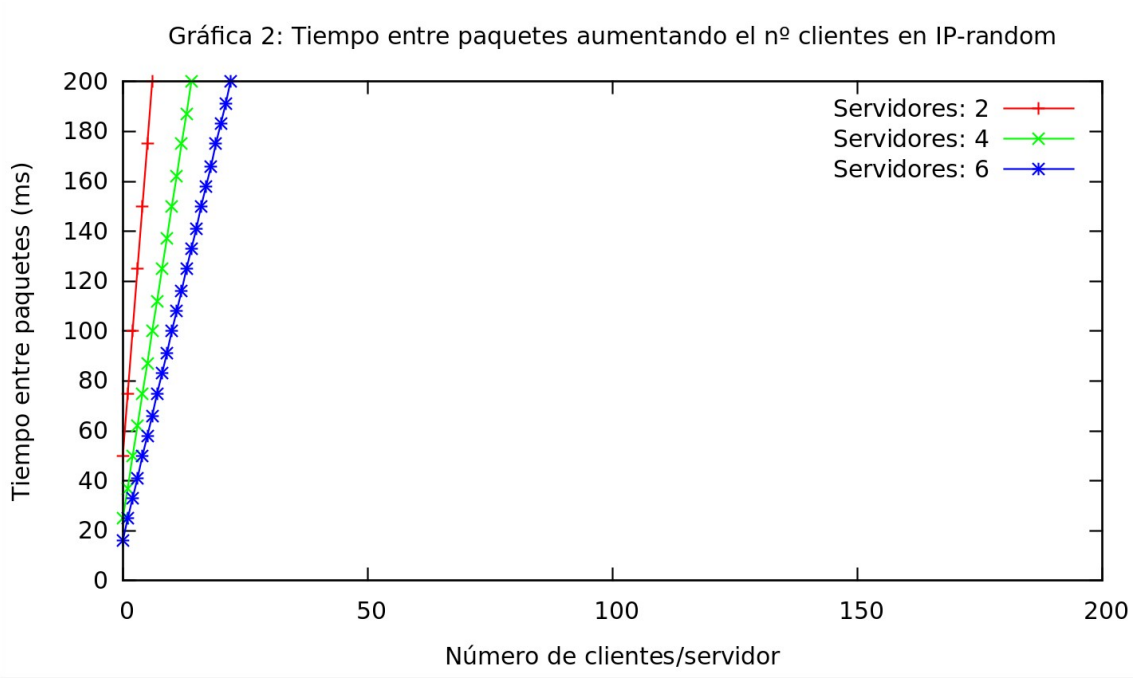
  Datos Iniciales
  - Clientes iniciales: 100
  - Tipo de balanceo: IP Random
  - Servidores iniciales: 2, 4 y 6
  - DataRate: 100Mbps
  - Delay: 5us

  Incrementos por punto
  - Clientes: 50

  En esta simulación vamos a observar dos gráficas: media de paquetes recibidos (*Gráfica 1*) y tiempo medio entre paquetes (*Gráfica 2*). En estas gráficas se van a representar 3 curvas, en las que tendremos 2, 4 y 6 servidores, cuyos valores son fijos, y donde vamos a utilizar un balanceo de carga de tipo IP random. Además se tiene un incremento de 50 clientes por cada punto.



Gráfica 1. Paquetes recibidos aumentando el nº de clientes en IP random



Gráfica 2. Tiempo entre paquetes aumentando el nº de clientes en IP random

A continuación procedemos a explicar brevemente los resultados de cada gráfica:

Gráfica 1:

En esta gráfica podemos observar como la evolución de los paquetes recibidos, en función del aumento del número de clientes según el número de servidores disponibles no varía conforme las gráficas mostradas con anterioridad.

Esto se debe a que independientemente del algoritmo utilizado, la media de paquetes recibidos en media por los servidores va a ser

similar, puesto que todos los paquetes van a ser repartidos entre ellos.

Gráfica 2:

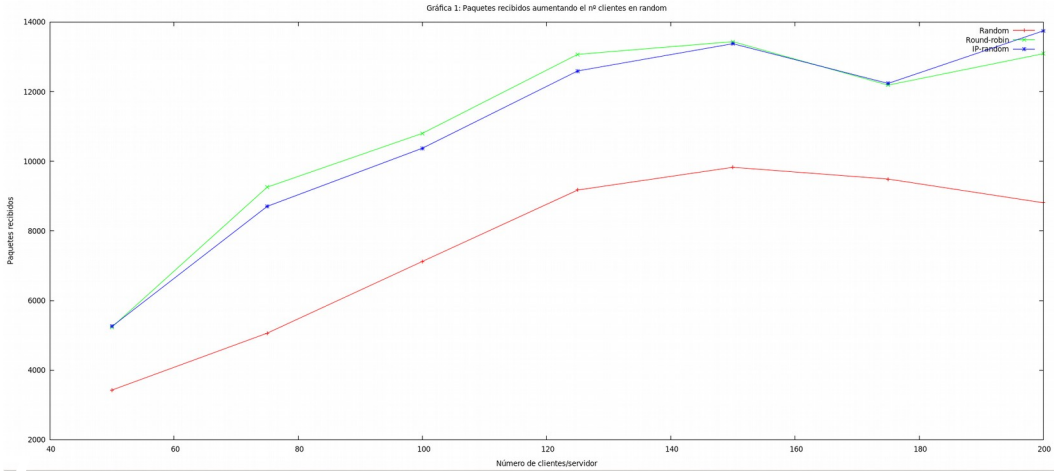
En esta gráfica podemos observar como varía considerablemente con el resto de algoritmos. Esto es debido a que este algoritmo sigue una distribución distinta a los algoritmos anteriores, basándose primero en la búsqueda de direcciones IP de clientes pares e impares para posteriormente hacer el reenvío. Debido a que el observador instalado en cada servidor no hace una distinción individual, sino que utiliza la media del conjunto el tiempo que aparece es mucho mayor y es por ello la razón del crecimiento tan repentino que se produce en esta gráfica.

- **Balanceo de carga con 4 servidores y variando el número de clientes**

  Datos Iniciales
  - Clientes iniciales: 100
  - Tipo de balanceo: Random, Round Robin e IP Random
  - Servidores iniciales: 4
  - DataRate: 100Mbps
  - Delay: 5us

  Incrementos por punto
  - Clientes: 50

  En esta simulación vamos a observar una gráfica que contiene la media de paquetes recibidos (*Gráfica 1*). En estas gráficas se van a representar 3 curvas, una por cada algoritmo de balanceo implementado, y en las que va a ir incrementándose el número de clientes en 50 por cada punto, dejando fijo el número de servidores en 4.



Gráfica 1. Paquetes recibidos aumentando el nº de clientes con los diferentes algoritmos de balanceo

En esta gráfica podemos observar la evolución de los paquetes recibidos, en función del aumento del número de clientes según el algoritmo de balanceo utilizado.
Al aumentar el número de clientes, como hay 4 servidores disponibles, las peticiones recibidas en media por cada servidor, van aumentando de forma considerable.
Además podemos observar como entre los diferentes algoritmos implementados, IP random es el algoritmo que recibe un mayor número de paquetes, por lo que posee menos pérdidas que el resto, siendo el algoritmo random el que menor número de paquetes recibe, produciendo así un mayor número de pérdidas.

6. Conclusiones

A raíz de los resultados obtenidos con anterioridad, podemos afirmar que los parámetros que para este caso concreto el algoritmo que mejor QoS mantiene es el algoritmo IP random, puesto que es el que mayor número de paquetes recibe con los mismos recursos que el resto. Por el contrario el peor algoritmo que se podría aplicar es el algoritmo random, puesto que posee un elevado número de pérdidas. El algoritmo round robin también es una buena opción para la implementación del servicio, no obstante posee un número de pérdidas ligeramente mayor al que contiene IP random.

6.1 Posibles mejoras

Pese a que este proyecto ha finalizado en este punto, somos conscientes que aún se podrían hacer algunas mejoras adicionales. Se nos han ocurrido una serie de mejoras que vamos a comentar brevemente a continuación.

6.1.1 Algoritmos de balanceo adicionales

En este proyecto hemos decidido optar por 3 algoritmos de balanceo diferentes para comprobar cuál de ellos es mejor opción. Obviamente no son los únicos algoritmos de balanceo que existen, por lo que se propone como mejora futura añadir nuevos algoritmos y comprobar si para el sistema propuesto mejoran a los algoritmos actuales, obteniendo así mejores estadísticas.

6.1.2 Creación de prioridades

Debido a la diferenciación de usuarios en plataformas que mezclan usuarios de pago y gratuitos, se podría implementar una diferenciación de clientes para darle mayor prioridad a aquellos clientes que se encuentren registrados en una BBDD. Para ello además habría que crear un sistema de colas que se encargue de hacer consultas a una BBDD y en base a ello sitúe los paquetes en una u otra prioridad.

6.1.3 Mejorar el manejo de errores

Para observar mejor el comportamiento que conlleva el programa, podemos crear un mayor número de trazas a diferente nivel y comprobar en todo momento en qué punto falla el programa y el motivo del fallo.

6.1.4 Balanceo de carga a nivel de red

En este proyecto el switch, mediante una consulta al controller, encamina las peticiones de los clientes a nivel de enlace a los servidores conectados a los puertos del mismo. No obstante estos puertos no son indefinidos, por lo que una posible mejora al proyecto sería que el switch tenga dos interfaces de red y en la consulta al controller en vez de indicar el puerto por el que enviar, indique la dirección IP privada del servidor al que enviarle la petición.

7. Anexos

7.1 Anexo A: Instalación de OpenFlow en ns-3

En este proyecto se va a hacer uso de la herramienta OpenFlow previamente explicada en detalle. Por lo que este apartado se va a centrar en la instalación de esta herramienta en la máquina virtual que se nos proporciona en la asignatura “Planificación y Simulación de Redes”.
Para ello vamos a seguir la guía que se nombra en la referencia 3. Se siguen los siguientes pasos:

1) Para no tener problemas con el fichero boost/static_assert.hpp tenemos que instalar la siguiente librería:

```
salas@ubuntu:~/ns-3.20$ sudo apt-get install libboost-all-dev
salas@ubuntu:~/ns-3.20$ sudo apt-get install mercurial
```

2) Obtenemos la carpeta OpenFlow.

```
salas@ubuntu:~/Descargas$ hg clone http://code.nsnam.org/openflow
destination directory: openflow
requesting all changes
adding changesets
adding manifests
adding file changes
added 3 changesets with 1031 changes to 1014 files
updating to branch default
919 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

3) Copiamos el directorio descargado anteriormente en la carpeta correspondiente y accedemos al mismo.

```
salas@ubuntu:~/Descargas$ cp -R openflow/ ../ns-3.20/ns3/
salas@ubuntu:~/Descargas$ cd ../ns-3.20/ns3/openflow/
```

4) Obtenemos la librería OpenFlow.

```
salas@ubuntu:~/ns-3.20/ns3/openflow$ ./waf configure
Checking for program gcc or cc           : /usr/bin/gcc
Checking for program cpp                 : /usr/bin/cpp
Checking for program ar                  : /usr/bin/ar
Checking for program ranlib              : /usr/bin/ranlib
Checking for gcc                         : ok
'configure' finished successfully (0.094s)
salas@ubuntu:~/ns-3.20/ns3/openflow$ ./waf build
Waf: Entering directory `/home/salas/ns-3.20/openflow/build'
Waf: Leaving directory `/home/salas/ns-3.20/openflow/build'
'build' finished successfully (0.056s)
```

5) Desde el directorio raíz de ns-3 configuramos OpenFlow en el sistema.

```
salas@ubuntu:~/ns-3.20/ns3/openflow$ cd ../../
salas@ubuntu:~/ns-3.20$ ./waf configure --enable-examples --enable-tests --with-
openflow=/home/salas/ns-3.20/openflow
```

6) Comprobamos que se ha activado correctamente OpenFlow y volvemos a reconstruir el sistema. Para comprobar que se ha activado la línea “NS-3 OpenFlow Integration” debe estar *enabled* como se muestra a continuación. Si no se ha activado OpenFlow no podremos continuar. El fallo posiblemente sea algo relacionado con el directorio que no lo estamos poniendo bien.

```
NS-3 OpenFlow Integration     : enabled
```

```
salas@ubuntu:~/ns-3.20$ ./waf build
```

7) Una vez ya tenemos instalado OpenFlow tenemos que modificar el fichero */home/dit/ns-3.20/src/openflow/model/openflow-interface.h*. Este fichero debe quedar de la siguiente forma:

```
// Include OFSI code
#include "ns3/simulator.h"
#include "ns3/log.h"
#include "ns3/net-device.h"
#include "ns3/packet.h"
#include "ns3/address.h"
#include "ns3/nstime.h"
#include "ns3/mac48-address.h"

#include <set>
#include <map>
#include <limits>

// Include main header and Vendor Extension files
#include "openflow/openflow.h"
#include "openflow/nicira-ext.h"
#include "openflow/ericsson-ext.h"
```

Tras la realización de estos pasos ya tenemos completamente configurado OpenFlow y ns-3 para poder realizar las distintas simulaciones que componen nuestro proyecto.

8. Referencias

[1] Página oficial de NS-3 - https://www.nsnam.org/

[2] Ejemplo de uso OpenFlow - http://code.nsnam.org/jpelkey3/openflow/summary

[3] Guía de instalación OpenFlow - https://www.nsnam.org/docs/models/html/openflow-switch.html

[4] Enlace a directorio GitHub del proyecto - https://github.com/carrodher/loadBalancer